METHOD AND APPARATUS FOR RESUMING EXECUTION OF A FAILED COMPUTER PROGRAM

FIELD OF THE INVENTION

The present invention generally relates to recovery from errors in computer programs, and more particularly to using alternative code to recover execution of a failed computer program.

BACKGROUND

Certain types of software errors are fatal to program execution. For example, a reference to a memory address that is beyond the address domain of a program will likely result in a fatal error. Certain timing or other transient conditions may also trigger fatal errors. While certain errors may be within the control of the software developer, the developer may be unable to guard against certain other errors in developing the software.

Program errors that are transient or timing-related are problematic to both the program user and to the program developer. From the user's point of view, failure of the program not only interrupts the task at hand, but may also result in lost work product. Reporting such transient errors is also difficult since the conditions required to reproduce the problem are likely to be unknown.

From the developer's point of view, much time may be spent trying to find the root cause of a problem where the root cause is external to the program. Furthermore, tracing the root cause of the problem may be difficult and time-consuming if there is there is scant information available for reproducing the problem.

A method and apparatus that address the aforementioned problems, as well as other related problems, are therefore desirable.

SUMMARY OF THE INVENTION

In various embodiments, a computer implemented method is provided for recovery from fatal program errors. A program is compiled using two compilers to generate first and second sets of object code. Checkpoints are identified in the program, and checkpoint code is generated for execution at the checkpoints. If execution of the first set of object code fails, checkpoint data is recovered and execution of the program is resumed using either the first or second set of object code. In one embodiment, the first set of object code is re-executed before trying the second set of object code. In another embodiment, the second set of object code automatically executed upon failure of the first set of object code.

It will be appreciated that various other embodiments are set forth in the Detailed Description and Claims which follow.

BRIEF DESCRIPTION OF THE DRAWINGS

Various aspects and advantages of the invention will become apparent upon review of the following detailed description and upon reference to the drawings in which:

FIG. 1 is a flowchart of a an example process for compiling a program in accordance with one embodiment of the invention;

FIG. 2 is a block diagram that illustrates an example data structure used in the management of checkpoints;

FIG. 3 is a block diagram that illustrates the code that results from compilation of a program in accordance with one embodiment of the invention; and

FIG. 4 is a flowchart of an example process for error recovery.

DETAILED DESCRIPTION

In various embodiments, the present invention provides a method and apparatus for generating alternative code that supports recovery from a fatal program error.

FIG. 1 is a flowchart of a an example process for compiling a program in accordance with one embodiment of the invention. The process generally entails generating two sets of object code segments using two different compilers or code generators. The second set of object code segments are available for execution in the event that the a fatal program error is encountered in executing the first set of segments. The segments of object code are delineated by checkpoints that are identified by the compiler.

At steps 202 and 204, the program source code is compiled using two different compilers, and two sets of object code that are functionally equivalent are created from the compilation. For example, the compilers may be different versions of the same compiler or compilers from different vendors. For making the two sets of object code interchangeable, checkpoints are identified and checkpoint code is generated for the two sets of object code at step 206. Known computer platforms allow multiple compilers to coexist and also allow inter-operation at the function/procedure call level.

In one embodiment, the checkpoints are identified and the code generated as described in the co-pending patent application entitled, "Compiler-based Checkpointing for Support of Error Recovery" by Ziegler et al. and filed on October 31, 2000, which has patent/application number ***** and attorney docket number 10001159, is commonly assigned to the assignee of the present invention, and the contents of which are incorporated herein by reference. Each segment of code is delineated by a checkpoint, as determined by the compiler. The checkpoints are points in

the code where the state of the program is stored so that execution can be recovered at the point in the program following the checkpoint. For example, convenient places for checkpoints are procedure boundaries.

At step 208, the code that performs the checkpointing is generated and combined with the intermediate code that was generated from the user's source code. Along with the checkpointing code, a data structure is created for storage of the checkpoint data at step 208.

FIG. 2 is a block diagram that illustrates an example data structure used in the management of checkpoints in accordance with one embodiment of the invention. To save storage space, two checkpoint data sets 260a and 260b are maintained.

Checkpoint data are alternately stored in checkpoint data sets 260a and 260b for consecutive checkpoints. For example, at time t1, checkpoint data set 260a references checkpoint 262 and checkpoint data set 260b references checkpoint 264. At time t2 after program execution completes checkpoint 266, checkpoint data set 260a references checkpoint 266, and checkpoint data set 260b references checkpoint 264.

Timestamps or commit flags may be used in alternative embodiments to indicate which of the checkpoint data sets is to be used in recovery. The timestamp scheme involves writing a timestamp to a checkpoint data set when the storage of state information in the checkpoint data set is complete. Thus, the later of the two timestamps indicates which of checkpoint data sets 260a or 260b is to be used in recovery. The commit flag scheme involves a flag that indicates which of checkpoint data sets 260a or 206b is to be used in recovery.

FIG. 3 is a block diagram that illustrates the code that results from compilation of a program in accordance with one embodiment of the invention. One purpose for compiling code in the manner taught herein is to enable recovery from fatal program errors. Block 102 represents program source code that is to be compiled and is comprised of *n* segments of source code. Checkpoints are used to delineate the multiple segments. A checkpoint is a location in the code at which execution can recommence should the program encounter a fatal error. At each checkpoint, the state of data elements used by the program are stored so that in the event of program failure the state information can be recovered and execution resumed immediately after the checkpoint from which the state was recovered. In various embodiments the checkpoints can be user-programmed or identified by the compiler using recognized techniques.

Compilation of the program source code results in program object code 104 that includes two sets of object segments, object segments 1-*n* and object segments 1-*n'*. The object segments in each set correspond to the source segments of program source code 102.

Each set of object segments is code that is generated in compiling the source code with different compilers or code generators. In other words, object segments 1-*n* are generated by a first compiler, and object segments 1'-*n'* are generated by a different compiler. If the program fails during execution of segment *i*, for example, then the state of the checkpoint data can be recovered from checkpoint that precedes segment *i* and execution can resume at segment *i'*.

FIG. 4 is a flowchart of an example process for error recovery in accordance with one embodiment of the present invention. The process generally entails recovering from

a fatal program error by restoring checkpoint data and resuming execution of the program. In resuming execution, the code of the first set of segments is retried. If the failure is repeated, execution of the program is resumed using the alternative set of segments.

The process begins when a program error has been detected by the operating system, for example. At step 304, checkpoint data is restored, and at step 306 the program counter is reset to the selected checkpoint. The segment of object code from the first set of segments is re-executed at step 308. If the program executes the segment without error, decision step 310 and step 312 illustrate that the program continues with execution of the segments from the first set.

If an error recurs in executing the segment of code from the first set, control is directed to decision step 314, which determines whether the alternative code should be tried. In one embodiment, the first set of segments of code may be re-executed a selected number of times before trying execution of the alternative code. Control is directed to step 316 when the decision is made to execute the alternative code.

At step 316, checkpoint data is restored, and at step 318, the address of the segment of object code from the second set is selected for execution. That is, the program counter is loaded with a program address of a segment in the second set. At step 320, the alternative segment of object code is executed, and execution of the segments of code from the second set continues at step 322.

Returning now to decision step 314, if the decision is made to not execute the alternative code, control is directed to decision step 324. Decision step 324 tests whether execution of the segment from the first set of segments should be attempted again. If so, control is

returned to step 304 to restore the checkpoint data and try again. Otherwise, the program is exited with an error.

The present invention is believed to be applicable to compilers for a variety of programming languages. Other aspects and embodiments of the present invention will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein. It is intended that the specification and illustrated embodiments be considered as examples only, with a true scope and spirit of the invention being indicated by the following claims.